

5

15

Systems such as the Internet typically are point-to-point (or unicast) systems in which a message is converted into a series of addressed packets which are routed from a source node through a plurality of routers to a destination node. In most communication protocols the packet includes a header which contains the addresses of the source and the destination nodes as well as a sequence number which specifies the packet's order in the message.

In general, these systems do not have the capability of broadcasting a message from a source node to all the other nodes in the network because such a capability is rarely of much use and could easily overload the network. However, there are situations where it is desirable for one node to communicate with some subset of all the nodes. For example, multi-party conferencing capability analogous to that found in the public telephone system and broadcasting to a limited number of nodes are of considerable interest to users of packet-switched networks. To satisfy such demands, packets destined for several recipients have been encapsulated in a unicast packet and forwarded from a source to a point in a network where the packets have

been replicated and forwarded on to all desired recipients. This technique is known as IP Multicasting and the network over which such packets are routed is referred to as the Multicast Backbone or MBONE. More recently, routers have become available which can route the multicast addresses (class D addresses) provided for in communication protocols such as TCP/IP and UDP/IP. A multicast address is essentially an address for a group of host computers who have indicated their desire to participate in that group. Thus, a multicast packet can be routed from a source node through a plurality of multicast routers (or mrouters) to one or more devices receiving the multicast packets. From there the packet is distributed to all the host computers that are members of the multicast group.

These techniques have been used to provide on the Internet audio and video conferencing as well as radio-like broadcasting to groups of interested parties. See, for example, K. Savetz et al. MBONE Multicasting Tomorrow's Internet (IDG Books WorldWide Inc., 1996).

Further details concerning technical aspects of multicasting may be found in the Internet documents Request for Comments (RFC) 1112 and 1458 which are reproduced at Appendices A and B of the Savetz book and in D. P. Brutaman et al., "MBONE provides Audio and Video Across the Internet," IEEE Computer, Vol. 27, No. 4, pp. 30-36 (April 1994), all of which are incorporated herein by reference.

Multimedia computer systems have become increasingly popular over the last several years due to their versatility and their interactive presentation style. A multimedia computer system can be defined as a computer system having a combination of video and audio outputs for presentation of audio-visual displays. A modem multimedia computer system typically includes one or more storage devices such as an optical drive, a CD-ROM, a hard drive, a videodisc, or an audiodisc, and audio and video data are typically stored on one or more of these mass storage devices. In some file formats the audio and video are interleaved together in a single file, while in other formats the audio and video data are stored in different files, many times on different storage media. Audio and video data for a multimedia

display may also be stored in separate computer systems that are networked together. In this instance, the computer system presenting the multimedia display would receive a portion of the necessary data from the other computer system via the network cabling.

5

Graphic images used in Windows multimedia applications can be created in either of two ways, these being bit-mapped images and vector-based images. Bit-mapped images comprise a plurality of picture elements (pixels) and are created by assigning a color to each pixel inside the image boundary. Most bit-mapped color images require one byte per pixel for storage, so large bit-mapped images create correspondingly large files. For example, a full-screen, 256-color image in 640-by-480-pixel VGA mode requires 307,200 bytes of storage, if the data is not compressed. Vector-based images are created by defining the end points, thickness, color, pattern and curvature of lines and solid objects comprised within the image. Thus, a vector-based image includes a definition which consists of a numerical representation of the coordinates of the object, referenced to a corner of the image.

10

15

20

25

Bit-mapped images are the most prevalent type of image storage format, and the most common bit-mapped-image file formats are as follows. A file format referred to as BMP is used for Windows bit-map files in 1-, 2-, 4-, 8-, and 24-bit color depths. BMP files contain a bit-map header that defines the size of the image, the number of color planes, the type of compression used (if any), and the palette used. The Windows DIB (device-independent bit-map) format is a variant of the BMP format that includes a color table defining the RGB (red green blue) values of the colors used. Other types of bit-map formats include the TIF (tagged image format file), the PCX (Zsoft Personal Computer Paintbrush Bitmap) file format, the GIF (graphics interchange file) format, and the TGA (Texas Instruments Graphic Architecture) file format.

30

The standard Windows format for bit-mapped images is a 256-color device-independent bit map (DIB) with a BMP (the Windows bit-mapped file format) or sometimes a DIB extension. The standard Windows format for vector-based images

000210-1988450

is referred to as WMF (Windows meta file).

Full-motion video implies that video images shown on the computer's screen simulate those of a television set with identical (30 frames-per-second) frame rates, and that these images are accompanied by high-quality stereo sound. A large amount of storage is required for high-resolution color images, not to mention a full-motion video sequence. For example, a single frame of NTSC video at 640-by-400-pixel resolution with 16-bit color requires 512K of data per frame. At 30 flames per second, over 15 Megabytes of data storage are required for each second of full motion video. Due to the large amount of storage required for full motion video, various types of video compression algorithms are used to reduce the amount of necessary storage. Video compression can be performed either in real-time, i.e., on the fly during video capture, or on the stored video file after the video data has been captured and stored on the media. In addition, different video compression methods exist for still graphic images and for full-motion video.

Examples of video data compression for still graphic images are RLE (run-length encoding) and JPEG (Joint Photographic Experts Group) compression. RLE is the standard compression method for Windows BMP and DIB files. The RLE compression method operates by testing for duplicated pixels in a single line of the bit map and stores the number of consecutive duplicate pixels rather than the data for the pixel itself. JPEG compression is a group of related standards that provide either lossless (no image quality degradation) or lossy (imperceptible to severe degradation) compression types. Although JPEG compression was designed for the compression of still images rather than video, several manufacturers supply JPEG compression adapter cards for motion video applications.

In contrast to compression algorithms for still images, most video compression algorithms are designed to compress full motion video. Video compression algorithms for motion video generally use a concept referred to as interframe compression, which involves storing only the differences between successive frames in the data file. Interframe compression begins by digitizing the entire image of a

5

15

20

25

5

10

15



15

20

In still yet another embodiment, a second loop may be created upon the beginning of a chapter during which information on a next chapter is retrieved.

5

10

15

20

25

25

30

Figure 8 illustrates a flowchart for affording a scheduler object adapted to facilitate the playback of an event simultaneously on a plurality of networked client apparatuses in accordance with one embodiment of the present invention;

5

Figure 9 is a flowchart delineating a method for identifying a plurality of events which are played back simultaneously on a plurality of networked client apparatuses in accordance with one embodiment of the present invention;

10 Figure 10 shows a flowchart delineating a technique for interfacing a plurality of different types of playback devices of client apparatuses which are networked to simultaneously playback an event in accordance with one embodiment of the present invention;

15 Figure 11 illustrates the manner in which a layer factory is created in accordance with one embodiment of the present invention;

Figure 12 illustrates the manner in which user requests are processed in accordance with one embodiment of the present invention;

20

Figures 13-16 illustrate various class/component diagrams in accordance with one embodiment of the present invention;

25 Figure 17 illustrates a logical sequence diagram in accordance with one embodiment of the present invention;

Figure 18 illustrates a logical sequence diagram that shows server side collaboration in accordance with one embodiment of the present invention; and

30 Figure 19 illustrates a logical sequence diagram showing client side collaboration in accordance with one embodiment of the present invention.

004488614.012000

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Figures 1-19 illustrate a system for synchronizing an event on a plurality of client apparatuses. Prior to use, an event is stored in memory on at least one of the client apparatuses. Such client apparatuses are adapted to be connected to a network along with a host computer(s). In operation, information is transmitted from the host computer to the at least one client apparatus utilizing the network. This information allows for the simultaneous and synchronous playback of the event on each of the client apparatuses.

In various embodiments, the client apparatuses may take the form of computers, televisions, stereos, home appliances, or any other types of devices. In one embodiment, the client apparatuses and the host computer each include a computer such as an IBM compatible computer, Apple Macintosh computer or UNIX based workstation.

A representative hardware environment is depicted in Figure 1, which illustrates a typical hardware configuration of a workstation in accordance with a preferred embodiment having a central processing unit 110, such as a microprocessor, and a number of other units interconnected via a system bus 112. The workstation shown in Figure 1 includes a Random Access Memory (RAM) 114, Read Only Memory (ROM) 116, an I/O adapter 118 for connecting peripheral devices such as disk storage units 120 (i.e. DVD playback device) to the bus 112, a user interface adapter 122 for connecting a keyboard 124, a mouse 126, a speaker 128, a microphone 132, and/or other user interface devices such as a touch screen (not shown) to the bus 112, communication adapter 134 for connecting the workstation to a communication network (e.g., a data processing network) and a display adapter 136 for connecting the bus 112 to a display device 138. The workstation typically has resident thereon an operating system such as the Microsoft Windows NT or Windows/95 Operating System (OS), the IBM OS/2 operating system, the MAC OS, or UNIX operating system. Those skilled in the art will appreciate that the present invention may also be implemented on platforms and operating systems other than those mentioned.

A preferred embodiment is written using JAVA, C, and the C++ language and utilizes object oriented programming methodology. Object oriented programming (OOP) has become increasingly used to develop complex applications. As OOP
5 moves toward the mainstream of software design and development, various software solutions require adaptation to make use of the benefits of OOP. A need exists for these principles of OOP to be applied to a messaging interface of an electronic messaging system such that a set of OOP classes and objects for the messaging interface can be provided.

10 OOP is a process of developing computer software using objects, including the steps of analyzing the problem, designing the system, and constructing the program. An object is a software package that contains both data and a collection of related structures and procedures. Since it contains both data and a collection of structures
15 and procedures, it can be visualized as a self-sufficient component that does not require other additional structures, procedures or data to perform its specific task. OOP, therefore, views a computer program as a collection of largely autonomous components, called objects, each of which is responsible for a specific task. This concept of packaging data, structures, and procedures together in one component or
20 module is called encapsulation.

In general, OOP components are reusable software modules which present an interface that conforms to an object model and which are accessed at run-time through a component integration architecture. A component integration architecture
25 is a set of architecture mechanisms which allow software modules in different process spaces to utilize each others capabilities or functions. This is generally done by assuming a common component object model on which to build the architecture. It is worthwhile to differentiate between an object and a class of objects at this point. An object is a single instance of the class of objects, which is often just called a
30 class. A class of objects can be viewed as a blueprint, from which many objects can be formed.

09483614-012000

5

10

20

25

30

behind the same name is called polymorphism and it greatly simplifies communication among objects.

With the concepts of composition-relationship, encapsulation, inheritance and polymorphism, an object can represent just about anything in the real world. In fact, one's logical perception of the reality is the only limit on determining the kinds of things that can become objects in object-oriented software. Some typical categories are as follows:

- Objects can represent physical objects, such as automobiles in a traffic-flow simulation, electrical components in a circuit-design program, countries in an economics model, or aircraft in an air-traffic-control system.
- Objects can represent elements of the computer-user environment such as windows, menus or graphics objects.
- An object can represent an inventory, such as a personnel file or a table of the latitudes and longitudes of cities.
- An object can represent user-defined data types such as time, angles, and complex numbers, or points on the plane.

With this enormous capability of an object to represent just about any logically separable matters, OOP allows the software developer to design and implement a computer program that is a model of some aspects of reality, whether that reality is a physical entity, a process, a system, or a composition of matter. Since the object can represent anything, the software developer can create an object which can be used as a component in a larger software project in the future.

If 90% of a new OOP software program consists of proven, existing components made from preexisting reusable objects, then only the remaining 10% of the new software project has to be written and tested from scratch. Since 90% already came from an inventory of extensively tested reusable objects, the potential domain from which an error could originate is 10% of the program. As a result, OOP enables software developers to build objects out of other, previously built objects.

This process closely resembles complex machinery being built out of assemblies and sub-assemblies. OOP technology, therefore, makes software engineering more like hardware engineering in that software is built from existing components, which are available to the developer as objects. All this adds up to an improved quality of the software as well as an increased speed of its development.

Programming languages are beginning to fully support the OOP principles, such as encapsulation, inheritance, polymorphism, and composition-relationship. With the advent of the C++ language, many commercial software developers have embraced OOP. C++ is an OOP language that offers a fast, machine-executable code. Furthermore, C++ is suitable for both commercial-application and systems-programming projects. For now, C++ appears to be the most popular choice among many OOP programmers, but there is a host of other OOP languages, such as Smalltalk, Common Lisp Object System (CLOS), and Eiffel. Additionally, OOP capabilities are being added to more traditional popular computer programming languages such as Pascal.

The benefits of object classes can be summarized, as follows:

- Objects and their corresponding classes break down complex programming problems into many smaller, simpler problems.
- Encapsulation enforces data abstraction through the organization of data into small, independent objects that can communicate with each other. Encapsulation protects the data in an object from accidental damage, but allows other objects to interact with that data by calling the object's member functions and structures.
- Subclassing and inheritance make it possible to extend and modify objects through deriving new kinds of objects from the standard classes available in the system. Thus, new capabilities are created without having to start from scratch.

- Polymorphism and multiple inheritance make it possible for different programmers to mix and match characteristics of many different classes and create specialized objects that can still work with related objects in predictable ways.
 - Class hierarchies and containment hierarchies provide a flexible mechanism for modeling real-world objects and the relationships among them.
 - Libraries of reusable classes are useful in many situations, but they also have some limitations. For example:
 - Complexity. In a complex system, the class hierarchies for related classes can become extremely confusing, with many dozens or even hundreds of classes.
 - Flow of control. A program written with the aid of class libraries is still responsible for the flow of control (i.e., it must control the interactions among all the objects created from a particular library). The programmer has to decide which functions to call at what times for which kinds of objects.
 - Duplication of effort. Although class libraries allow programmers to use and reuse many small pieces of code, each programmer puts those pieces together in a different way. Two different programmers can use the same set of class libraries to write two programs that do exactly the same thing but whose internal structure (i.e., design) may be quite different, depending on hundreds of small decisions each programmer makes along the way. Inevitably, similar pieces of code end up doing similar things in slightly different ways and do not work as well together as they should.
 - Class libraries are very flexible. As programs grow more complex, more programmers are forced to reinvent basic solutions to basic problems over and over again. A relatively new extension of the class library concept is to have a framework of class libraries. This framework is more complex and consists of significant collections of collaborating classes that capture both the small scale patterns and major mechanisms that implement the common requirements and design in a specific application domain. They were first developed to free application

programmers from the chores involved in displaying menus, windows, dialog boxes, and other standard user interface elements for personal computers.

Frameworks also represent a change in the way programmers think about the
5 interaction between the code they write and code written by others. In the early days
of procedural programming, the programmer called libraries provided by the
operating system to perform certain tasks, but basically the program executed down
the page from start to finish, and the programmer was solely responsible for the flow
of control. This was appropriate for printing out paychecks, calculating a
10 mathematical table, or solving other problems with a program that executed in just
one way.

The development of graphical user interfaces began to turn this procedural
programming arrangement inside out. These interfaces allow the user, rather than
15 program logic, to drive the program and decide when certain actions should be
performed. Today, most personal computer software accomplishes this by means of
an event loop which monitors the mouse, keyboard, and other sources of external
events and calls the appropriate parts of the programmer's code according to actions
that the user performs. The programmer no longer determines the order in which
20 events occur. Instead, a program is divided into separate pieces that are called at
unpredictable times and in an unpredictable order. By relinquishing control in this
way to users, the developer creates a program that is much easier to use.
Nevertheless, individual pieces of the program written by the developer still call
libraries provided by the operating system to accomplish certain tasks, and the
25 programmer must still determine the flow of control within each piece after it's
called by the event loop. Application code still "sits on top of" the system.

Even event loop programs require programmers to write a lot of code that should not
need to be written separately for every application. The concept of an application
30 framework carries the event loop concept further. Instead of dealing with all the
nuts and bolts of constructing basic menus, windows, and dialog boxes and then
making these things all work together, programmers using application frameworks

000210-4T988460

start with working application code and basic user interface elements in place. Subsequently, they build from there by replacing some of the generic capabilities of the framework with the specific capabilities of the intended application.

Application frameworks reduce the total amount of code that a programmer has to write from scratch. However, because the framework is really a generic application that displays windows, supports copy and paste, and so on, the programmer can also relinquish control to a greater degree than event loop programs permit. The framework code takes care of almost all event handling and flow of control, and the programmer's code is called only when the framework needs it (e.g., to create or manipulate a proprietary data structure).

A programmer writing a framework program not only relinquishes control to the user (as is also true for event loop programs), but also relinquishes the detailed flow of control within the program to the framework. This approach allows the creation of more complex systems that work together in interesting ways, as opposed to isolated programs, having custom code, being created over and over again for similar problems.

20 Thus, as is explained above, a framework basically is a collection of cooperating
classes that make up a reusable design solution for a given problem domain. It
typically includes objects that provide default behavior (e.g., for menus and
windows), and programmers use it by inheriting some of that default behavior and
overriding other behavior so that the framework calls application code at the
25 appropriate times.

There are three main differences between frameworks and class libraries:

- Behavior versus protocol. Class libraries are essentially collections of behaviors that you can call when you want those individual behaviors in your program. A framework, on the other hand, provides not only behavior but also the protocol or set of rules that govern the ways in which behaviors can

be combined, including rules for what a programmer is supposed to provide versus what the framework provides.

- Call versus override. With a class library, the code the programmer instantiates objects and calls their member functions. It's possible to instantiate and call objects in the same way with a framework (i.e., to treat the framework as a class library), but to take full advantage of a framework's reusable design, a programmer typically writes code that overrides and is called by the framework. The framework manages the flow of control among its objects. Writing a program involves dividing responsibilities among the various pieces of software that are called by the framework rather than specifying how the different pieces should work together.
- Implementation versus design. With class libraries, programmers reuse only implementations, whereas with frameworks, they reuse design. A framework embodies the way a family of related programs or pieces of software work. It represents a generic design solution that can be adapted to a variety of specific problems in a given domain. For example, a single framework can embody the way a user interface works, even though two different user interfaces created with the same framework might solve quite different interface problems.

Thus, through the development of frameworks for solutions to various problems and programming tasks, significant reductions in the design and development effort for software can be achieved. A preferred embodiment of the invention utilizes HyperText Markup Language (HTML) to implement documents on the Internet together with a general-purpose secure communication protocol for a transport medium between the client and the Newco. HTTP or other protocols could be readily substituted for HTML without undue experimentation. Information on these products is available in T. Berners-Lee, D. Connolly, "RFC 1866: Hypertext Markup Language - 2.0" (Nov. 1995); and R. Fielding, H. Frystyk, T. Berners-Lee, J. Gettys and J.C. Mogul, "Hypertext Transfer Protocol -- HTTP/1.1: HTTP Working Group Internet Draft" (May 2, 1996). HTML is a simple data format used to create hypertext documents that are portable from one platform to another. HTML

documents are SGML documents with generic semantics that are appropriate for representing information from a wide range of domains. HTML has been in use by the World-Wide Web global information initiative since 1990. HTML is an application of ISO Standard 8879; 1986 Information Processing Text and Office Systems; Standard Generalized Markup Language (SGML).

To date, Web development tools have been limited in their ability to create dynamic Web applications which span from client to server and interoperate with existing computing resources. Until recently, HTML has been the dominant technology used in development of Web-based solutions. However, HTML has proven to be inadequate in the following areas:

- Poor performance;
- Restricted user interface capabilities;
- Can only produce static Web pages;
- Lack of interoperability with existing applications and data; and
- Inability to scale.

Sun Microsystem's Java language solves many of the client-side problems by:

- Improving performance on the client side;
- Enabling the creation of dynamic, real-time Web applications; and
- Providing the ability to create a wide variety of user interface components.

With Java, developers can create robust User Interface (UI) components. Custom "widgets" (e.g., real-time stock tickers, animated icons, etc.) can be created, and client-side performance is improved. Unlike HTML, Java supports the notion of client-side validation, offloading appropriate processing onto the client for improved performance. Dynamic, real-time Web pages can be created. Using the above-mentioned custom UI components, dynamic Web pages can also be created.

Sun's Java language has emerged as an industry-recognized language for "programming the Internet." Sun defines Java as: "a simple, object-oriented,

distributed, interpreted, robust, secure, architecture-neutral, portable, high-performance, multithreaded, dynamic, buzzword-compliant, general-purpose programming language. Java supports programming for the Internet in the form of platform-independent Java applets." Java applets are small, specialized applications that comply with Sun's Java Application Programming Interface (API) allowing developers to add "interactive content" to Web documents (e.g., simple animations, page adornments, basic games, etc.). Applets execute within a Java-compatible browser (e.g., Netscape Navigator) by copying code from the server to client. From a language standpoint, Java's core feature set is based on C++. Sun's Java literature states that Java is basically, "C++ with extensions from Objective C for more dynamic method resolution."

Another technology that provides similar function to JAVA is provided by Microsoft and ActiveX Technologies, to give developers and Web designers wherewithal to build dynamic content for the Internet and personal computers. ActiveX includes tools for developing animation, 3-D virtual reality, video and other multimedia content. The tools use Internet standards, work on multiple platforms, and are being supported by over 100 companies. The group's building blocks are called ActiveX Controls, small, fast components that enable developers to embed parts of software in hypertext markup language (HTML) pages. ActiveX Controls work with a variety of programming languages including Microsoft Visual C++, Borland Delphi, Microsoft Visual Basic programming system and, in the future, Microsoft's development tool for Java, code named "Jakarta." ActiveX Technologies also includes ActiveX Server Framework, allowing developers to create server applications. One of ordinary skill in the art readily recognizes that ActiveX could be substituted for JAVA without undue experimentation to practice the invention.

Synchronization Overview

Figure 2 illustrates a flowchart delineating a method for synchronizing an event on a plurality of client apparatuses. First, in operation 200, an event is stored in memory on at least one of the client apparatuses. In various embodiments, the memory may

take the form of an electromagnetic medium, or any type of optical storage device, i.e. CD-audio. In a primary aspect of the present invention, the memory includes a digital video disc (DVD) (audio or video). Further, for reasons that will soon become apparent, the information includes chapter information associated with the DVD. In such embodiment where the memory is portable, the user may be required to purchase the memory, i.e. DVD, in order to participate in a synchronized event, thus increasing the sale of DVD's.

It should be noted that the event need not be necessarily stored in memory on all of the client apparatuses, but rather stored on one or some of the client apparatuses and streamed to the remaining client apparatuses at variant rates. This may be feasibly accomplished if the client apparatus(es) containing the stored event has a high-bandwidth connection with the remaining client apparatuses. For example, the client apparatus(es) containing the stored event may include a server that has a connection to a plurality of televisions via a cable network, i.e. WEBTV. Similar functionality may be achieved via a broadcast medium. The present invention is thus flexible by having an ability to host user events and corporative events.

In one embodiment, the event includes a video and audio presentation such as movie, a concert, and/or a theatrical event. It should be noted, however, that the event may included any recording capable of being played back for entertainment, education, informative or other similar purposes.

In use, the client apparatuses and a host computer are adapted to be connected to a network. Such network may include a wide, local or any other type of communication network. For example, a wide area network such as the Internet may be employed which operates using TCP/IP or IPX protocols.

In operation 202, information is transmitted from the host computer to the appropriate client apparatuses utilizing the network. This information allows for the simultaneous and synchronous playback of the event on each of the client apparatuses. In one embodiment, the information may also include a start time when

the playback of the event is to begin on each of the client apparatuses. Further, an ending time may be included when the playback of the event is to end on each of the client apparatuses. Still yet, "play" command information may be sent to the client apparatuses at the start time. As an option, input may be received from the user, and
5 used to alter the playback of the event. The host server, or synchronization server, can also control various streams of a variant rate and different hardware associated with those streams.

The present invention thus has the ability to synchronize video playback for one or
10 multiple (thousands) users from one or multiple physical locations, and to synchronize with external video, audio and/or data streams.

Users of the present invention are at multiple physical locations and host servers may also be at different locations. The present invention is thus a scalable system
15 which is capable of servicing an unlimited number of users. Since the content is local to the user machine, no high network bandwidth is required.

History Download Capabilities

20 Figure 3 illustrates a flowchart delineating a method for storing synchronization information for subsequent playback of an event. Initially, in operation 300, an event is stored in memory on at least one of the client apparatuses, as set forth earlier. These client apparatuses are adapted to be connected to a network along with a host computer during use.

25 In operation 302, information is stored on the host computer(s) for allowing the simultaneous playback of the event on each of the client apparatuses. In one embodiment, the information may include a history and data associated with the synchronous playback. In particular, the history may include any overlaid
30 material(as will be described hereinafter in greater detail), any specific commands affecting the playback of the information, or any other type of general information, i.e. start time, end time, etc.

09488614-012000

5

Overlay Synchronization

10 Figure 4 illustrates a flowchart setting forth a method for providing overlays during a synchronized event on a plurality of client apparatuses or any other source. First, in operation 400, a plurality of client apparatuses are connected via a network. In operation 402, an event may be simultaneously played back on the client apparatuses utilizing the network, as set forth earlier.

15 During the playback of the event, visual and/or audio material may also be overlaid on the event based on input received from at least one of the client apparatuses. See operation 404. This may be accomplished by transmitting the overlay material from one of the client apparatuses to the host computer or any other server, and

20 multicasting the same to the remaining client apparatuses.

25

30

It should be noted that various bi-directional communication may be enabled for
5 allowing data to travel to and from the server. For instance, the playback of the event on
the client apparatuses may be altered in any feasible way based on input from a user.

Figure 5 illustrates a flow diagram for delayed synchronization of an event on a plurality of client apparatuses. First, in operation 500, a plurality of client apparatuses are connected via a network and an event is stored in memory on the client apparatuses. The event is then simultaneously played back on the client apparatuses utilizing the network, as set forth earlier. Note operation 502.

During the simultaneous playback, a request may be received from one of the client apparatuses for that particular to be included in the synchronized event, as set forth in operation 504. This request may be received after the synchronized event has already begun while it is still playing. Further, the request may be submitted via a site on a network, i.e. website.

25 This allows the simultaneous playback of the event on the requesting client
apparatus.

different audiences. Also language and cultural differences can be taken into account. Still yet, the present invention may be adapted to address users on different hardware platforms (MAC, PC, set-top boxes). This may be accomplished by

identifying the user using a cookie, a user profile which is identified by way of a log in, or a Burn Cut Area (BCA) of the disc.

An example setting forth details relating to identifying DVDs will now be set forth.

5 First, a content owner (such as studio) requests use of the BCA on their DVDs. Based on request, the replicator (examples include WAMO, Panasonic, Nimbus, Technicolor, Pioneer, Crest) adds unique BCA number to every DVD. Adding BCA number to each DVD requires a special (YAG) laser. This may be the very last step in the manufacturing process. The BCA numbers for a specific DVD must then be entered
10 into InterActual's BCA database. Information to track includes: DVD title, i.e. "Lost in Space"; BCA #/range, i.e. 12345687890; and Shipping Packaging/Tracking Container, i.e. Box 52221 to Hollywood Video.

After the BCA number is added to the DVDs, the DVDs are packaging/boxed for
15 distribution to either the Distributor or the Retailer. It should be noted that many companies take multiple forms, so the replicator and distributor may be one in the same. Also, some retailers are large/important enough to get shipments directly from replicator. The way in which the DVDs are packaging/shipped is very important because one must track the BCA numbers to actual shipping containers
20 (box, etc.). Therefore tracking information must also be added to the BCA database.

If packaged DVDs are then sent to distributor, the distributor also has mechanisms, i.e. scanners, input device, and monitoring devices, in place for tracking based on their distribution. For example, Deluxe may receive a "package" of 100,000 copies
25 of "Lost in Space." However, the distributor ships 10,000 to Retailer A and 5,000 to Retailer B. The distributor should be able to "input" retailer A and B's distribution information into the system. Ideally, this becomes a seamless/automated process.

Once the DVDs reach the retailer (either from the replicator or distributor), then
30 DVDs may be further divided and distributed to local stores/outlets. In such a situation, the retailer should be able to automatically "track" distribution of these DVDs through to their stores. Over time, all three entities (replicator, distributor,

000279" 4988460
09488614" 012000

5

10

15

20

25

30

Using ISAPI, you create a Dynamic Link Library (DLL) application file that can run as part of the Hypertext Transport Protocol (HTTP) application's process and address space. The DLL files are loaded into the computer when HTTP is started and remain there as long as they are needed; they don't have to be located and read into storage as frequently as a CGI application.

Existing CGI applications can be converted into ISAPI application DLLs without having to rewrite their logic. However, they do need to be written to be thread-safe so that a single instance of the DLL can serve multiple users.

A special kind of ISAPI DLL is called an ISAPI filter, which can be designated to receive control for every HTTP request. One can create an ISAPI filter for encryption or decryption, for logging, for request screening, or for other purposes.

One can write ISAPI server extension DLLs (ISAs) that can be loaded and called by the HTTP server. Users can fill out forms and click a submit button to send data to a Web server and invoke an ISA, which can process the information to provide custom content or store it in a database. Web server extensions can use information in a database to build Web pages dynamically, and then send them to the client computers to be displayed. An application can add other custom functionality and provide data to the client using HTTP and HTML.

One can write an ISAPI filter. The filter is also a DLL that runs on an ISAPI-enabled HTTP server. The filter registers for notification of events such as logging on or URL mapping. When the selected events occur, the filter is called, and one can monitor and change the data (on its way from the server to the client or vice versa). ISAPI filters can be used to provide custom encryption or compression schemes, or additional authentication methods.

Both server extensions and filters run in the process space of the Web server, providing an efficient way to extend the server's capabilities.

Overall Component Design

The various functional components of the software associated with the present invention will now be set forth. Such components include a Java/JavaScript Component, Synchronizer Component, LayerImpl Component, Business Layer
5 Component, Configuration Manager Component, and DBConnect Component.

Java/JavaScript Component

Figure 6 illustrates a flow diagram for providing information on a synchronized
10 event on a plurality of client apparatuses in accordance with one embodiment of the present invention. First, in operation 600, a plurality of client apparatuses are connected via a network, as set forth earlier. Next, an application program is embedded on a site on the network in operation 602. Such application program may take the form of a JAVA applet, and the site may include a website on the Internet.

15 In use, information is requested from a server on the network utilizing the application program. See operation 604. Such information relates to an event to be played back simultaneously on the client apparatuses and may include general information such as a start and stop time of the event, or more specific information
20 about the event itself.

In response to such request, a script is received for displaying the information. Note operation 606. The script may take any form such as Perl, REXX (on IBM mainframes), and Tcl/Tk, and preferably includes a JAVAscript.

25 In one embodiment of the present invention, the JAVA applet may be further adapted to send a request to retrieve command information from the server for use with a playback device of one of the client apparatuses. The commands may be adapted to playback the event on the playback device simultaneous with the
30 playback of the event on the remaining client apparatuses. Further, the commands may include a start time when the playback of the event is to begin on each of the client apparatuses.

The JAVA applets and JAVAscript are used to communicate with the playback device of the client apparatuses. In one embodiment, the playback device includes a PCFriendly™ video player manufactured by Interactual®.

5

The Java applet is embedded within a web page and uses HTTP protocol to communicate to the synchronization server. The applet could request event information from the server, and display it to the user via JavaScript. The applet could also send a "*BroadcastVideoEvent*" request to retrieve DVD commands that can be passed to the video component, as set forth hereinabove.

10

Synchronizer Component

Figure 7 illustrates a method for creating a synchronizer object in order to playback an event simultaneously on a plurality of client apparatuses. The synchronizer object is portion of the software that actually implements the synchronization procedure. First, in operation 700, a request is received utilizing a network for viewing an event. Next, the request is queued in memory in operation 702.

15

In response to the request, in operation 704, an object is created which is adapted to playback the event on a client apparatus simultaneous with the playback of the event on the remaining client apparatuses upon the receipt of an activation signal. As an option, the activation signal may be provided using a clock of the client apparatus, or located at a different location, i.e. server. To accomplish this, the object identifies a start time when the playback of the event is to begin on each of the client apparatuses.

20

25

In operation 706, the object is sent to one of the client apparatuses utilizing the network for being stored therein. In accordance with a primary aspect of the present invention, the object may be adapted to playback the event which is stored in memory of the client apparatus. This may be accomplished by activating a digital video disc (DVD) player.

30

000270" 4988460

In summary, when the Synchronizer component receives a "*BroadcastVideoEvent*" from the applet, it then places the request in the thread queue for processing. To process a request, the thread creates a "call back" object, if one does not exist for this event. The thread then adds the request to the "call back" object queue. This "call back" object will be invoked when it is time to play the DVD. The Synchronizer component creates a Call Back COM object, *LayerSink*. The Synchronizer component is also responsible for creating the *LayerFactory* interface which will be set forth hereinafter in greater detail.

LayerImpl Component

Figure 8 illustrates a flowchart for affording a scheduler object adapted to facilitate the playback of an event simultaneously on a plurality of networked client apparatuses. The present method ensures that critical information is tracked during the synchronization of the event. Such critical information not only ensures proper synchronization, but also enables various peripheral features.

First, in operation 800, various values are determined including a current time, a start time when an event is to start, and a stop time when the event is to end. Thereafter, a length of the event is calculated based on the start time and the stop time in operation 802. As an option, the current time is determined by querying a clock of one of the client apparatuses.

If any portion of the length of the event takes place during a predetermined threshold period, a command is stored in memory in operation 804. The command may be adapted to automatically begin playing back the event at the start time. In one embodiment, the threshold period includes the time the users can be queued before the event. As an option, chapter information may be stored in the memory if any portion of the length of the event takes place during the predetermined threshold period. This allows the command to automatically begin playing back the event at a predetermined chapter.

The “call back” object (*LayerSink*) is thus responsible for creating and communicating with the *LayerImpl* component. The *LayerImpl* component acts as a scheduler, determining when to issue commands to the user.

The following is an enumerated summary of the steps the component uses to
20 determine when the users will receive the DVD commands:

1. Retrieves the current time, and the time the event starts and ends.
2. Calculates the length of the event.
3. If the event is within a threshold period (i.e. the time users can be queued before the event), then store the first DVD command in memory. Also, store the Chapter information in memory.
4. Create a loop that processes request until the event has completed.
5. In the loop, calculate the lapsed time of the event.
6. In the loop, retrieve the next chapter information.
7. Create another loop that will loop until time for the next chapter to be played.
8. When the next chapter is ready to play, send the command that was retrieved from the Chapter table.

Business Layer Component

Figure 9 is a flowchart delineating a method for identifying a plurality of events which are played back simultaneously on a plurality of networked client apparatuses. This features is important since a host server may be synchronizing more than one event at once, or during overlapping times. Such events must therefore be distinguished.

10 First, in operation 900, a plurality of events are stored in memory on a plurality of client apparatuses. Each of the events is assigned a unique identifier which is stored in the memory.

In operation 902, the client apparatuses are adapted to be coupled to a host computer via a network, as set forth hereinabove. In operation 904, the identifier of the event which is stored in the memory of the client apparatuses is then retrieved utilizing the network. Such identifier is subsequently compared with an identifier of a scheduled event, as set forth in operation 906. If the comparison renders a match, the playback of the event is begun on the appropriate client apparatuses. Note operation 908.

CbusinessLayer thus differentiates events by the disk and location ids, uploaded by the client to guarantee backwards compatibility. As set forth earlier, late arrivals can always re-sync with the event.

25 Configuration Manager Component

Figure 10 shows a flowchart delineating a technique for identifying playback devices of a plurality of client apparatuses which are networked to simultaneously playback an event. The present technique is important since the playback devices of the various client apparatuses may differ in make and model. Thus, different commands are required therefor.

operation 1002. Such table may be located at the host server, or at any other location such as the client apparatuses.

10 of the event on each of the remaining client apparatuses.

15 requested, the “Synchronizer” table is queried for the host type. From that point forward, the scheduler would know from whom to receive data.

DBConnect Component

20 This component is responsible for communicating with the Synchronizer tables, and for providing access methods for the retrieved data. All interaction from the tables is on a read-only basis. The *LayerImpl* component communicates with this component to retrieve DVD commands and event information.

25 Even though current implementation may be based on a Microsoft platform, hard dependencies on Microsoft or any other 3rd-party development tools may be avoided. To address such issues, the following considerations may be made throughout the code:

30 MFC specific code may be avoided. Instead, STL may be used. ATL and/or MFC code may be encapsulated into separate classes and portioned from the rest of the code. Class implementations may use aggregation pattern to delegate business logic

5 Figures 11 and 12 illustrate the order of events among the various components of the present invention. In particular, Figure 11 illustrates the manner in which a layer factory is created. As shown, an event is first checked in a database server after which a business layer is created in a WEB server in a manner set forth hereinabove. The foregoing components are then created. Figure 12 illustrates the manner in which user requests are processed. As shown, communication is afforded with the video player on the client machine by means of JAVAscript and JAVA applets. The WEB server, in turn, communicates DVD commands to the video player via the JAVA applets, and also interfaces the database server via the various components thereof which were set forth hereinabove.

15

20

Hosted Real Time Players

25

30

Clients may maintain connections throughout the event. This allows the host to send a various number of commands to the client of the event. Although the spiral disconnects users once a PLAY command has been issued, the Synchronizer class (which will be set forth later) adds each connection to a Thread Pool. This pool of connections can be left open during the life of the event.

Logging Participants

Each request may be logged into the database to provide a reference for the future.

DVD Positioning

As an option, connections may be pooled to allow the synchronization server to direct consumer's machines to the certain locations throughout the entire event.

Synchronization events in alternate spirals may be defined as a combination of play from location event and the actual event. This way, one describes each event in the unambiguous way on the client side and synchronizes it with the server. For example, a situation may be considered where one fast forwards after a movie is played for 15 min and thereafter plays the scene in the movie. In such situation, one has to submit the information to the client player, indicating that it (player) has to start time play from 15 min into the movie and fast-forward to the certain location. A better way would be to analyze what is the next event after fast forwarding occurred and perform a combination for the play from location and next event. This design would require significant changes to the client infrastructure, including *video object*, *remoteagent* and *provider* and should be taken into consideration in any alternate client design.

Classes/Component Diagrams

Figures 13-16 illustrate various class/component diagrams. In particular, Figures 13-16 illustrate a Synchronizer Class Diagram 1300, LayerImpl Class Diagram 1400,

Business Layer Class Diagram 1500, and DBConnect Class Diagram 1600, respectively.

Sequence Diagrams

Figure 17 illustrates a logical sequence diagram 1700. As shown, when the server receives a user request, it analyzes the authentication information of the request (date/time, disc id, user id, and BCA number) and the appropriate synchronization event stored in the database. The database contains an *event start threshold* value measured in milliseconds. This threshold defines the amount of time prior to an event that a consumer is eligible to “connect” for the start of the event.

If the date/time of the user request lies within the event start threshold, the user is put into wait queue and receive the appropriate data when the time elapses. Note steps 1,2,3,5,6,7 of the Logical Sequence diagram. Otherwise, a message is sent informing the user when the event will occur. Note step 4 of the Logical Sequence diagram.

Server side collaboration diagram

Figure 18 illustrates a logical sequence diagram 1800 that shows server side collaboration. As shown, server ISAPI extension receives a BroadcastVideoEvents request. It calls *IA_BusinessServer* via *BeginProcess*, to retrieve configuration information. Configuration information contains a playback connector. Playback connector identifies whether the server will have to communicate with a reference player or will it perform playback from the database.

At step 6, ISAPI extension will call *IA_BusinessServer CompareTime* method and based on the results will send to the user a predefined web page indicating to retry later or return control to the web server, notifying it (web server) to keep the connection open. At this point connection is pooled and will be processed by the *IA_BusinessServer* at a time of the event.

Client Collaboration Diagram

Figure 19 illustrates a logical sequence diagram 1900 showing client side
5 collaboration in accordance with one embodiment of the present invention.

Classes/Interfaces Definition

Definitions of one embodiment of the various classes associated with the software which
10 implements the present invention will now be set forth.

Class Applet1

Purpose:

15

This is the class that implements the applet. The browser will use it to bootstrap our
applet.

Responsibilities:

20

- Request a BroadCastVideo event and to gather event status information.

Collaborations:

25 BroadCastEvent, CITIEncrypt

Base class and implemented interfaces:

Javax.Applet

30

Public interface:

000270"4T988460
09488614-012000

5 Post-conditions: None.

Post-conditions: **None.**

Post-conditions: None.

Post-conditions: **None.**

30 Post-conditions: None.

getStartTimeHr Returns the time the event starts in Hours.

Post-conditions: None.

GetStartTimeDay Returns the time the event starts in days.

10 Post-conditions: None.

Post-conditions: None.

Post-conditions: None.

Post-conditions: None.

Parameters: void

Pre-conditions: None.

Post-conditions: None.

getServerTime: Returns the servers current time <SS:MM:HH:DD:MM:YYYY>.

5 Return type: String

Parameters: void

Pre-conditions: None.

Post-conditions: None.

10 **getServerTimeSec:** Returns the servers current in seconds.

Return type: String

Parameters: void

Pre-conditions: None.

Post-conditions: None.

15

getServerTimeMin: Returns the servers current in minutes.

Return type: String

Parameters: void

Pre-conditions: None.

20 Post-conditions: None.

getServerTimeHr: Returns the servers current in hours.

Return type: String

Parameters: void

25 Pre-conditions: None.

Post-conditions: None.

getServerTimeDay: Returns the servers current in day.

Return type: String

30 Parameters: void

Pre-conditions: None.

Post-conditions: None.

000210"4T988460

getServerTimeMnth: Returns the servers current in month.

Return type: String

Parameters: void

5 Pre-conditions: None.

Post-conditions: None.

getServerTimeYr: Returns the servers current in year.

Return type: String

10 Parameters: void

Pre-conditions: None.

Post-conditions: None.

startProc: Calls the ISAPIs "ServerInfo" method.

15 Return type: void

Parameters: String disk id, String location id

Pre-conditions: None.

Post-conditions: None.

20 **msgEvent:** Calls BroadCastEvent applet.

Return type: void

Parameters: void

Pre-conditions: None.

Post-conditions: None.

25

Class BroadCastEvent

Purpose:

30 This is the class that invokes the Synchronizer.

Responsibilities:

09463614-012000

- Sets the JavaScript with the command returned from the server.

Collaborations:

5

CITIEncrypt

Base class and implemented interfaces:

10 Java.Thread

Class CDBConnect

Purpose:

15

This is the class provides a public interface for components to request information from the DB tables.

Responsibilities:

20

- Opens the database and Synchronizer, Chapter_Disk tables.
- Queries the Synchronizer by the specified disk id and location id.
- Queries the Chapter_Disk by disk id.
- Provides the next chapter that is scheduled to play.
- 25 ▪ Queries the Decoder_Capabilities table to determine if the requested player is time or chapter play.

Collaborations:

30 DBSyncSet

DBReferenceSet

CDBChapterSet

000270"4T988460
09488614-012000

CDecoderCapabilities

Base class and implemented interfaces:

5 *Public interface:*

Get_NextChapter: Returns the next chapter to play.

Return type: String

Parameters: long time, long title, BSTR Chapter

10 Pre-conditions: None.

Post-conditions: None.

chkEvent: Checks if an event is scheduled for the disk and location id.

Return type: String

15 Parameters: long time, long title, BSTR Chapter

Pre-conditions: None.

Post-conditions: None.

get_initialDVDCommand: Returns the first DVD command to play.

20 Return type: String

Parameters: BSTR &

Pre-conditions: None.

Post-conditions: None.

25 **get_nextDVDCommand:** Returns the next DVD command to play.

Return type: String

Parameters: BSTR &

Pre-conditions: None.

Post-conditions: None.

30

decoderArray: Returns an array of decoder types.

Return type: String

0002F0"4T988460

Parameters: long **, long **

Pre-conditions: None.

Post-conditions: None.

5 Class CConfigMgrImpl

Purpose:

10 This is the class provides a public interface for components to determine the type of reference player hosting the event.

Responsibilities:

- Opens the database and Synchronizer, Chapter_Disk tables.
- 15 ▪ Queries the Synchronizer by the specified disk id and location id.
- Stores the reference player type.

Collaborations:

20 CConfigMgrRecSet

Base class and implemented interfaces:

Public interface:

25

get_hostType: Returns the reference player host type.

Return type: String

Parameters: short

Pre-conditions: None.

30 Post-conditions: None.

Class threadFunctor

09483614-012000

Purpose:

This class provides a threading model that classes can use to derive.

5

Responsibilities:

- Calls the CreateEvent function, which opens a named or unnamed event objec.
- 10 ▪ Calls _beginthread, which creates a thread begins execution of a routine at start_address. The routine at start_address must use the __cdecl calling convention and should have no return value. When the thread returns from that routine, it is terminated automatically.
- 15 ▪ Calls the WaitForSingleObject function, which checks the current state of the specified object. If the object's state is nonsignaled, the calling thread enters an efficient wait state.
- Calls the ResetEvent function, which sets the state of the specified event object to nonsignaled.
- 20 ▪ The state of an event object remains nonsignaled until it is explicitly set to signaled by the SetEvent or PulseEvent function.

Collaborations:

CConfigMgrRecSet

25

Base class and implemented interfaces:

Public interface:

30 **start:** Starts the thread.

Return type: void

Parameters: void

Post-conditions: None.

5 Return type: void

Pre-conditions: None.

Post-conditions: **None.**

10 Class isapithread

Purpose:

This creates an ISAPI thread.

Responsibilities:

- Adds a request to a vector.
- Creates the sink object.
- 20 ▪ Stores the request into sink object.
- Sends the time information to JavaScript.

Collaborations:

```
25  LayerSink
    factorySink
```

Base class and implemented interfaces:

30 threadFunctor

Public interface:

Post-conditions: **None.**

Post-conditions: **None.**

Base class and implemented interfaces:

Public interface:

construct: Stores a layerSink object..

Return type: void

5 Parameters: void

Pre-conditions: None.

Post-conditions: None.

notifyCreateLayer: Responsible for creating a "businessLayerProp".

10 Return type: void

Parameters: BSTR, BSTR, DATE, DATE, LONG

Pre-conditions: None.

Post-conditions: None.

15 Class layerSink

Purpose:

layerSink represents a sink interface and stores a queue of requests. It creates a
20 connection point object.

This call back object, allows asynchronously processing.

Responsibilities:

- 25
 - Acts as the client sink object.
 - Sends the results to the user
 - Creates the "BusinessLayer" and makes it a connection point object.
 - Closes the users connection.
 - Creates a Factory interface by calling "createFactory".
- 30
 - Creates a connection point for the factory.
 - Stores the LayerSink in the FactorySink object.

000210"4F988460

- Creates a connection point (call back) by calling AtlAdvise, between the connection point container and the client sink object. This allows the client to receive events.
- Calls the connectable objects "getServerLayer". This method fires an event to the clients sink object.
- Create a business layer,
- Store the request in its vector.
- Release the Sink Object (client)
- Calls AtlUnadvise to terminates the ability of the client to receive events.

Collaborations:

Base class and implemented interfaces:

Public interface:

construct: Creates a connection point.

Return type: void

Parameters: void

Pre-conditions: None.

Post-conditions: None.

addRequest: Adds the request to its vector.

Return type: void

Parameters: BSTR, BSTR, DATE, DATE, LONG

Pre-conditions: None.

Post-conditions: None.

createBusinessLayer: Creates a business layer. Create the connection point .

Return type: void

Parameters: businessLayerProp &

Pre-conditions: None.

Post-conditions: None.

updatetime: This call back function translates the time and sends the command to the user.

5 Return type: void

Parameters: long,long

Pre-conditions: None.

Post-conditions: None.

10 Class CBusinessLayer

Purpose:

Creates a layerthread object. This object is responsible for providing access methods,
15 which provide event information.

Responsibilities:

- 20 ▪ The "Synchronizers" createBusinessLayer method creates a class object from the "IBusinessLayer" interface. <The class object is part of the LayerImpl project>
- 25 ▪ The BusinesLayers class object <m_ilyer> calls its "Initialize" method. <Note: m_ilyer is the connection point object. It identifies the "Sink Interface".>
- It then calls the "Initialize" method of the connection point.
- The "Initialize" method then calls the "ChkValidEvent" method, which then creates a layerthread object.

Collaborations:

30

CBusinessLayer

layerthread

000270" 47998460

Base class and implemented interfaces:

Public interface:

5

Initialize: Calls the "ChkValidEvent" method which kicks of a layer thread.

Return type: void

Parameters: void

Pre-conditions: None.

10 Post-conditions: None.

Class layerthread

Purpose:

15

This object acts as a scheduler, processing request from its queue.

Responsibilities:

- 20
- Send DVD commands to the user.
 - "Syncs" up late comers to the events.

Collaborations:

25 CBusinessLayer
CDBConnect

Base class and implemented interfaces:

30 *Public interface:*

startThread: Processes requests from the queue

00020"4T989h60
09488614.012000

Parameters: void

Post-conditions: **None.**

Class CLayerFactory

10 This object manages businesslayer objects. Business layer objects communicate with
the reference player and notify the user which DVD command to play.

```

15      ▪ Send DVD commands to the user.
      ▪ “Syncs” up late comers to the events.
      ▪ This object Implements the IID_LayerFactory interface.
      ▪ This COM object is the servers Connectable Point object.
      ▪ This server object supports connections to sink interfaces. These sink
20  interfaces reside on the client side and are equivalent to the "call back"
      functions in Windows.

```

```
25  CBusinessLayer
    CDBConnect
```

30 *Public interface:*

getServerLayer: "Fires" an event to create a business layer with the properties retrieved from the pipe object.

Return type: void

Parameters: void

5 Pre-conditions: None.

Post-conditions: None.

put_set_layer: call the "CLayerFactoryImpl" add() method. Supplying the "businesslayer" object.

10 This will added to shared memory queue and written to a file.

Return type: void

Parameters: void

Pre-conditions: None.

Post-conditions: None.

15

FinalConstruct: Calls the "CLayerFactoryImpl" FinalConstruct COM class object.

Return type: void

Parameters: void

Pre-conditions: None.

20 Post-conditions: None.

Although only a few embodiments of the present invention have been described in detail herein, it should be understood that the present invention may be embodied in many other specific forms without departing from the spirit or scope of the invention. Therefore, the present examples and embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope of the appended claims.

000210"4T989460